Interactive Programs in Haskell
(Software Workshop Haskell)

Antoni Diller

8 November 2010

```
-- This file is in:  ard/teaching/workshop/public/active.hs

capitalises :: String -> String
capitalises = map toUpper


before x = takeWhile (/= x)
after x  = tail . dropWhile (/= x)


-- capitalises input with echo, but still no sensible way to terminate
-- run with command: interact capitalises


capitalises1 :: String -> String
capitalises1 input = line ++ "\n" ++ map toUpper line
                          ++ "\n" ++ capitalises input'
                    where line   = before '\n' input
                          input' = after '\n' input


-- interactive program, with echo, that cubes what is input,
-- but still no sensible way to terminate
-- run with command: interact cubes


cubes :: String -> String
cubes input = line ++ "\n" ++ (toString (cube (fromString line)))
                   ++ "\n" ++ cubes input'
              where line   = before '\n' input
                    input' = after '\n' input
```

```
-- eg, fromString "853" = 853
fromString :: String -> Int
fromString = dec . map digitToInt


-- eg, dec [8, 5, 3] = 853
dec :: [Int] -> Int
dec [i] = i
dec is = 10 * dec (init is) + last is

-- eg, toString 853 = "853"
toString :: Int -> String
toString = map intToDigit . spread

-- eg, spread 853 = [8, 5, 3]
spread :: Int -> [Int]
spread i
  | i < 10  = [i]
  | i >= 10 = spread (div i 10) ++ [mod i 10]

cube i = i * i * i

-- capitalises2 terminates with EOL input

capitalises2 :: String -> String
capitalises2 = takeWhile (/= '\n') . map toUpper

-- capitalises alphabetic input, cubes numeric input,
-- ends with two EOL characters, error otherwise
-- input echoes only after EOL character input
-- run with command: interact capcube

capcube :: String -> String
capcube input
  | line == [] = []
  | and (map isAlpha line) = line ++ "\n" ++ map toUpper line
                                  ++ "\n" ++ capcube input'
  | and (map isDigit line) = line ++ "\n" ++ (toString (cube (fromString line)))
                                  ++ "\n" ++ capcube input'
  | otherwise              = line ++ "\n" ++ "Not valid input"
                                  ++ "\n" ++ capcube input'
                             where line   = before '\n' input
                                   input' = after '\n' input
```

```
-- capitalises alphabetic input, cubes numeric input,
-- ends with two EOL characters, error otherwise
-- input echoes as soon as a key is pressed
-- run with command: interact capcubes


capcubes :: String -> String
capcubes input
  = line ++ "\n" ++ alt line input'
    where
    line   = before '\n' input
    input' = after '\n' input
    alt xs ys
      | xs == [] = []
      | and (map isAlpha xs) = map toUpper xs    ++ "\n" ++ capcubes ys
      | and (map isDigit xs) = (toString (cube (fromString xs)))
                                                   ++ "\n" ++ capcubes ys
      | otherwise            = "Not valid input" ++ "\n" ++ capcubes ys


type StCo = String -> String


read1 :: String -> (String -> StCo) -> StCo
read1 msg g input = msg ++ line ++ "\n" ++ g line input'
                    where line   = before '\n' input
                          input' = after  '\n' input


cubing :: String -> String
cubing = read1 [] cube'
         where
         cube' ll ii = toString (cube (fromString ll)) ++ "\n" ++ cubing ii


read2 :: (String,String) -> ((String,String) -> StCo) -> StCo
read2 (msg1, msg2) g = read1 msg1 g1
                       where g1 line1 = read1 msg2 g2
                                        where g2 line2 = g (line1, line2)
```

```
write :: String -> StCo -> StCo
write msg g input = msg ++ g input


end _ = ""


enter :: a -> b -> [(a,b)] -> [(a,b)]
enter k v t = (k, v) : t


lookup1 :: Eq a => [(a,String)] -> a -> String
lookup1 t k
  | vs /= []  = head vs
  | otherwise = "No entry"
                where vs = [w | (c, w) <- t, c == k]

newtable = []


table :: [(String,String)] -> StCo
table t = read1 "Command: " tcommand
          where
          tcommand "enter"  = read2 ("Key: ", "Value: ") tenter
          tcommand "lookup" = read1 "Key: " tlookup
          tcommand "end"    = write "Exit program\n" end
          tcommand _        = write "Unknown command\n\n" (table t)
          tenter (k, v)     = write "\n\n" (table (enter k v t))
          tlookup k         = write (lookup1 t k ++ "\n\n") (table t)
```